Darpa/Navy Contract No. N00014-92-J-1809

ControlShell: A Real-Time Software Framework

Quarterly Progress Report — January - March, 1992

P.I.'s: Prof. J.C. Latombe, Prof. R.H. Cannon, Jr, Dr. S.A. Schneider

Executive Summary

We are creating a new paradigm for building and maintaining complex real-time software systems for the control of moving mechanical systems. This objective is being met through the *simultaneous* development of both a powerful software environment and cogent motion planning and control capabilities. Our research concentrates on three key areas:

- Building an innovative, powerful real-time software framework,
- Implementing new distributed control architectures for intelligent mechanical systems, and
- Developing distribution architectures and new algorithms for the computationally "hard" motion planning and direction problem.

Perhaps more importantly, we are working on the *vertical integration* of these technologies into a powerful, working system. It is only through this coordinated, cooperative approach that a truly revolutionary, usable architecture can result.

Summary of Progress

This section highlights some of our achievements for this first quarter. During this period, we have:

- Completed a first-level design of and an initial implementation of the Network Data Delivery Service (NDDS).
- Developed a high level module system specification for our cooperating robot system, and identified the major interfaces.
- Identified several axes for distributing path planning software in an on-line architecture.
- Designed and implemented a new landmark-based mobile robot planning method.
- Defined the layout of a software toolkit to efficiently develop new navigation systems.
- Started application support for projects in composite layup and ultra precision machining.

Our research is progressing according to schedule.

Chapter 1

Introduction

The goal of this research project is to build a new paradigm for building and maintaining complex real-time software systems for the control of moving mechanical systems. This objective is being met through the *simultaneous* development of both a powerful software environment and cogent motion planning and control capabilities. Our research concentrates on three areas:

- Building an innovative, powerful real-time software development environment,
- Implementing a new distributed control architecture, and using it to deftly control and coordinate real mechanical systems, and
- Developing a computation distribution architecture, and using it to build on-line motion planning and direction capabilities.

We believe that no technology can be successful unless proven experimentally. We are thus validating our research by direct application in several disparate, real-world settings.

This concurrent development of system framework, sophisticated motion planning and control software, and real applications insures a high-quality architectural design. It will also embed, in *reusable* components, fundamental new contributions to the science of intelligent motion planning and control systems. Researchers from our three organizations, the Stanford Aerospace Robotics Laboratory (ARL), the Stanford Computer Science Robotics Laboratory (CSRL), and Real-Time Innovations, Inc. (RTI) have teamed to cooperate intimately and directly to achieve this goal. The potential for advanced technology transfer represented by this cooperative, vertically-integrated approach is unprecedented.

Framework Development This research builds on an object-oriented tool set for real-time software system programming known as *ControlShell*. It provides a series of execution and data interchange mechanisms that form a framework for building real-time applications. These mechanisms are specifically designed to allow a component-based approach to real-time software generation and management. By defining a set of interface specifications for inter-module interaction, ControlShell provides a common platform that is the basis for real-time code exchange and reuse.

Our research is adding fundamental new capabilities, including network-extensible data flow control and a graphical CASE environment.

Distributed Control Architecture This research combines the high-level motion planning component developed by the previous effort with a deft control system for a complex multi-armed robot. The emphasis of this effort is on building interfaces between modules that permit a complex real-time system to run as an interconnected set of distributed modules. To drive this work, we are building a dual-arm cooperative robot system that will be able to respond to high-level user input, create sophisticated motion and task-level plans, and execute them in real time. The system will be able to effect simple assemblies while reacting to changing environmental conditions. It combines a world modelling system, real-time vision, task and path planners, an intuitive graphical user interface, an on-line simulator, and sophisticated control algorithms.

Computation Distribution Architecture This research thrust addresses the issues arising when computationally complex algorithms are embedded in a real-time framework. To illustrate these issues we are considering two particular problem domains: *object manipulation by autonomous multi-arm robots* and *navigation of multiple autonomous mobile robots in an incompletely known environment.* These two problems raise a number of generic issues directly related to the general theme of our research: motion planning is provably a computationally hard problem and its outcomes, motion plans, are executed in a dynamic world where various sorts of contingencies may exist.

The ultimate goals of our investigation are to both provide real-time controllers with on-line motion reactive planning capabilities and to build experimental robotic systems demonstrating such capabilities. Moreover, in accomplishing this goal, we expect to identify general guidelines for embedding a capability requiring provably complex computations into a real-time framework.

Chapter 2

ControlShell Framework Development

This section describes our progress in developing the ControlShell framework and underlying architecture. Two fundamental extensions to ControlShell are being pursued:

- Distributed information sharing paradigms, by Gerardo Pardo-Castellote and Stan Schneider.
- Graphical Computer Aided Software Engineering (CASE) environments, by Stan Schneider and Vince Chen.

We identified the need for facile information sharing models as critical to the project's development and are therefore addressing it first. This quarter, work has concentrated on our new distributed event-driven information sharing paradigm, the Network Data Delivery Service (NDDS).

2.1 Distributed Information Sharing Paradigms: NDDS

With the addition of network-extensible data flow control, ControlShell will become an easily scalable architecture. Integrating distributed data flow directly into the superstructure will enable transparent-yet-controlled access to system data by modules at all levels, greatly facilitating complex system design. Run-time binding of data on remote processors will also enable the direct support of systems with distributed components or requiring remote teleoperation, such as co-operating teams of mobile robots and remotely operated vehicles. This extension represents a fundamental advance in the state-of-the-art of real-time software architectures.

Real-Time Data Flow Characterization Many control systems are naturally distributed. This is due to the fact that often they are composed of several physically distributed modules:

sensors, command, control and monitoring modules. In order to achieve a common task, these modules need to share information.

These information sharing needs are common to many other application environments such as databases, distributed computing, parallel computing, transaction systems, etc. However, distributed control applications have some unique requirements:

- Data transactions in control applications are often time-critical. To be useful for control purposes data must get from the producer to the consumer with minimum delay.
- The time when data was produced is often critical for many control applications. This is required for things like filtering, prediction etc.
- Data dependencies are critical. Control applications are often data driven. For example a control-command depends on the sensor data, a collision-avoidance plan depends on a sensor update indicating the presence of a new obstacle, the output of a filter depends on its input etc. As a consequence, there is often the need to synchronize computation to the arrival of new data.
- Most of the data flow is repetitive in nature. This is true of sensor readings, motor commands etc. Data loss is often not critical; sending data is an idempotent operation, since new updates just replace old values. This property suggests that considerable overhead can be avoided by setting up a naturally repetitive data transfer paradigm.
- There are often multiple sources of what may be considered the same data item. For example, a robot command might be generated by a planner module as well as a tele-operation module. In the same way there can be many data-sinks. A robot and a simulator are both sinks of "command-data." This networks of data producers and consumers aren't known in advance and may change dynamically at run time as new modules are swapped in while the system is in operation.
- Data requirements are ubiquitous and unpredictable. It is often very difficult to know what data will be required by other modules. For instance, force-level measurements—normally used only by a low-level controller—may be required by a very high-level task planner in the future. The architecture should support these types of data flow. Thus, vital data should be accessible throughout the system.
- Most data flow can be anonymous. Producers of the sensor readings can usually be unaware of who is reading them. Consumers may not care where the data they use came from. Since it is not essential, hiding this information increases modularity by allowing the data sources and sinks to change transparently.

These requirements are sufficiently unique to deserve special treatment. To address all this needs we have developed the Network Data Delivery Service (NDDS): a new information sharing paradigm.

The NDDS system is built around the model of information producers (sources) and consumers (sinks). Producers register a set of object instances that they will produce, unaware of prospective consumers and "produce" the data at their own discretion. Consumers "subscribe" to updates of any object instance they require without concern for who is producing them. In this sense the NDDS is a "subscription-based" model.

Using subscriptions allows us to drastically reduce the overhead required by a client-server architecture. Occasional subscription requests, at low bandwidth, replace numerous high-bandwidth client requests. Latency is also reduced, as the outgoing request message time is eliminated.

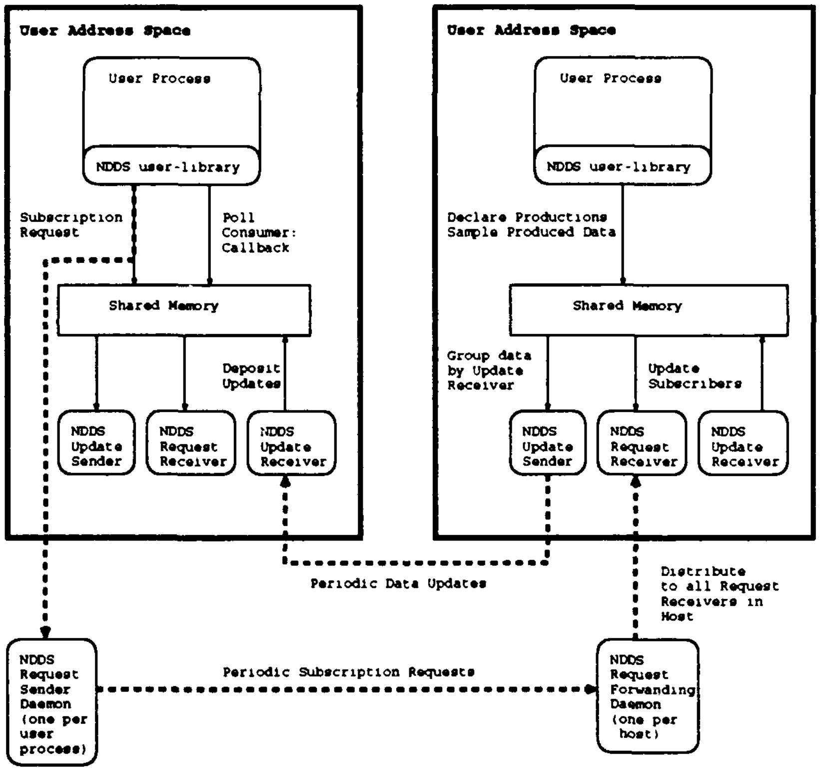

Figure 2.1: **Threads involved in the Network Data Delivery Service (NDDS).**

There are 6 distinct threads involved in NDDS. The user thread (which uses the NDDS library) and five daemons. The Request-Sender, Request-Forwarding and Request-Receiver daemons are responsible for distributing and processing subscription requests. The Update-Sender and Update-Receiver daemons handle the distribution of updates.

During this first contract quarter, we have completed the design of the NDDS and implemented an initial version of it.

Figure 2.1 illustrates the software architecture. We have completed several tests of this architecture and distributed data across three different computer architectures: the Sun workstations in the Aerospace Robotics Laboratory , our real-time computer system (a VME bus based multiprocessor system running the VxWorks operating system) that control our robots and the DEC workstations in the Stanford Computer Science Robotics Laboratory . All our tests have been successful.

Chapter 3

Distributed Control Architectures and Interfaces

This section covers our research in software architectures, communication protocols and interfaces that will advance the state-of-the-art in the prototyping-development-testing cycle of high-performance distributed control systems. These interfaces will be implemented within the framework described in Chapter 2. The results of this research will be applied to the vertical integration of planning and control and demonstrated by executing a set of challenging tasks on our two-armed robot system.

There are three main thrusts to this research:

- Development of inter-module interfaces for distributed control systems, by Gerardo Pardo-Castellote.
- Development of a control methodology capable of executing high-level commands, by Gerardo Pardo-Castellote, Tsai-Yen Li, and Yotto Koga.
- Hardware development and experimental verification, by Gerardo Pardo-Castellote and Gad Shelef.

Most of our efforts this quarter were directed toward framework development, especially NDDS (see Chapter 2. A solid framework design is required as a basis for the intermodule interface and control methodology development.

3.1 Inter-Module Interfaces

We have identified the need for three main interfaces: The task-level interface, the world model interface and the robot interface. Figure 3.1 illustrates the overall system architecture and role of these interfaces. These interfaces will be built on top of NDDS. This will allow us to distribute our computation and maintain an "open-systems" approach that will allow more modules to be added later. Moreover, the existence of these interfaces will allow us to create "standard" component modules to provide well-defined functions such as path planning.

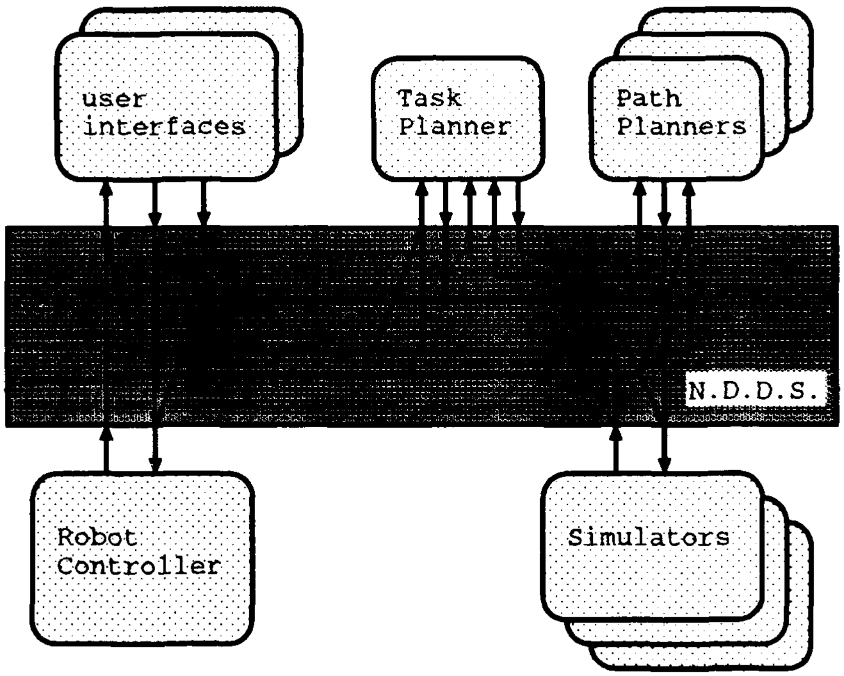

Figure 3.1: **System Building Blocks.**

There are 5 distinct functional block that compose our distributed control system. The user interface is responsible for providing graphical feedback to the user on the system state and allows the user to issue commands as different levels (from task commands down to tele-operation). The task planner is responsible for breaking the commanded tasks into lower level robot commands that the robot can execute. The planner provides path planning services to both the task planner and the robot. The simulator serves several functions: it allows end-to-end testing of all the system components except for the robot itself. It provides a tool for validation system models by comparing its behavior with that of the real robot and, provides visual feedback to the user. These modules are interconnected using the Network Data-Delivery Service .

3.2 Hardware Development and Experiments

We have identified an earlier need for enhancements to our two-armed robot platform that we expected. In our initial proposals we had this upgrade scheduled for the second year. However, we have found it difficult to perform significant tasks within the limited workspace of the present manipulators. Also, the large diameter wrist assemblies greatly hamper fine motion. So, we have decided that even our early our experimental demonstrations will benefit from the added functionality and extended workspace that the upgrade will provide. In view of this, we plan to complete the first demonstration (later this year) with the current hardware while pursuing a new design in parallel. This will require us to do the design and build the hardware upgrades during this first contract year.

Starting at the end of the first year we have scheduled an ambitious set of demanding demonstrations that will serve both as a guide and a test-bed for our research. This demonstrations will form the basis of our design-iterate cycle, essential to generating well-tested, robust solutions that will be made available to the rest of the community.

Our real-time computer system needs to be upgraded to support the new functionality that we are developing. We plan to purchase two new real-time processors (Motorola MVME-167) based on Motorola's 68040 micro-processor to replace two MVME-133 (which are based on Motorola's 68020 chip). This will provide at least a factor of three performance improvement.

Chapter 4

On-Line Computation Distribution Architectures

This research addresses technical issues arising when computationally complex algorithms are embedded in a real-time framework. To illustrate these issues we are considering two particular problem domains: *object manipulation by autonomous multi-arm robots* and *navigation of multiple autonomous mobile robots in an incompletely known environment.* These two problems raise a number of generic issues directly related to the general theme of our research: motion planning is provably a computationally hard problem and its outcomes, motion plans, are executed in a dynamic world where various sorts of contingencies may occur.

The ultimate goal of our investigation is to both provide real-time controllers with on-line motion reactive planning capabilities and build experimental robotic systems demonstrating such capabilities. Moreover, in accomplishing this goal, we expect to identify general guidelines for embedding a capability requiring provably complex computations into a real-time framework.

During this period, our work has focused on the following areas:

1. Distribution of Path Planning, by Tsai-Yen Li.
2. Parallelization of Path Planning, by Lydia Kavraki and Tsai-Yen Li.
3. New Methods for Fast Path Planning, by Tsai-Yen Li.
4. Optimal-Time Manipulation Multi-Arm Planning, by Yotto Koga.
5. Experiments in Manipulation Planning, by Tsai-Yen Li and Yotto Koga.
6. Landmark-Based Mobile Robot Navigation, by Anthony Lazanas.
7. Mobile Robot Navigation Toolkits, by Mark Yim and David Zhu,

8. Multi-Mobile Robot Simulator, by David Zhu.

Areas 1 through 5 are mainly related to the first problem domain, i.e., *object manipulation by autonomous multi-arm robots.*

Areas 6 through 8 are mainly related to the second problem domain, i.e., *navigation of multiple autonomous mobile robots in an incompletely known environment.*

Participating Ph.D. Students: Lydia Kavraki, Yotto Koga, Anthony Lazanas, Tsai-Yen Li, Mark Yim.

Participating Staff: Randall Wilson, David Zhu.

4.1 Distribution of Path Planning

Path planning is now a relatively well understood problem, but it is provably a computationally expensive one. Therefore, in order to allow path planning techniques to be exploited in a real-time framework, we develop and explore the underlying concepts of a new software architecture allowing planning algorithms to be distributed over available resources. Unlike previously proposed distributed architectures, our architecture seeks to distribute computation along a variety of dimensions.

In a preliminary step, we have identified a collection of interesting axes along which path planning computation can be usefully distributed. We describe them below. Other axes may be identified in the future.

- **Distribution over interconnected parallel processors:** This is the most obvious axis for distributing software. A given planning method may be parallelized so that it is concurrently executed on several processors, in order to get significant speedups. However, some planning methods may be more parallelizable than others. Furthermore, different sorts of parallelisms can be considered, e.g., coarse grain vs. fine grain. In the beginning, we plan to focus our work mainly on the Randomized Path Planner (RPP) developed by our group over the past few years,[1] since it is the fastest of its kind and it is a good candidate for various forms of parallelization. We plan to re-implement this planner on interconnected parallel processors and to conduct experimental analysis to guide future work in this area. Moreover, parallelization is also an important domain of study for the other distribution axes. If multiple powerful processors are available, all the other axes of distribution described below naturally yield some simple form of coarse-grain parallelization, although this parallelization is not necessary.
- **Distribution over problem approximations:** The most important parameters influencing the running time of a motion planning method are usually well-known, e.g., the number of

[1] RPP was developed under DARPA contract DAAA21-89-C0002.

degrees of freedom, the dimension of the workspace, the number and the degree of the algebraic equations describing the objects' boundaries (i.e., the complexity of the boundaries). Given a planning problem, the running time can often be reduced by one or several orders of magnitude by approximating the problem into a simplified, but still realistic one, before running the planner. The approximation may, for example, consist of simplifying the shape of the objects and/or freezing some degrees of freedom of the robots. If the simplification is conservative, a solution of the simplified problem, if any, is also a solution of the original problem. However, there is no unique set of parameters which will work best in all situations. The tradeoff is typically between completeness and efficiency. We plan to explore the possibility of changing the problem approximation dynamically and/or, if several parallel processors are available, solving the problem with different fixed approximations in parallel to achieve the statistically shortest running time.

- **Distribution over planning methods:** Several practical motion planning methods have been developed over the last few years. Their operational characteristics (completeness, running time) are often different, and there does not seem to exist a planner that is always better than all the others. Even a single planner, e.g. a potential-field-based planner, can be tuned into a collection of different planners by simply changing some of its internal parameters (e.g., the potential function). Therefore, several planners may be used to solve the same problem. A judicious choice among these planners for every planning problem may result in a substantial improvement of efficiency. Again, if several processors are available, one may try to run different planners concurrently to solve the same problem. For example, one may run an incomplete planner and a complete planner in parallel; while the first planner may return a path in a small amount of time if it is lucky (but may also just fail if it is not so lucky), the second is guaranteed to ultimately generate a path (if one exists). There is probably no ideal planning method, each having pathological cases where it does not work so well. Certain planning methods may be suitable for certain environments, but not for others. We plan to explore techniques to automatically select the most promising methods over which planning can be distributed.

- **Distribution of commitment over time:** A path is usually an overly committed plan to achieve a goal, and many alternative paths are equally acceptable. Committing a robot to a single path has the drawback that any event may make the path invalid, requiring that another path be replanned. Rather, the controller may do better by distributing commitment over time. For instance, in a first phase of processing, it may generate a "channel", i.e. a sort of tunnel containing a continuous infinity of paths for a robot; in a second phase, it may guide the robot through the channel using a simple potential field method. Generating a channel may not be faster, nor significantly slower, than generating a single path, but it may often avoid the relatively high cost of replanning paths. On the other hand, it may also allow the controller to ignore small obstacles in the first phase (in which case channel generating might become significantly faster than path generation), provided that these small objects are taken into account in the potential function used in the second phase. Provided that the ignored objects be small enough with respect to the size of the channel, there is very little chance for the machine to get stuck somewhere in the channel. If this occurs, there would still be the possibility of performing some local re-planning. Distributing commitment over time requires the planners to generate richer information about possible paths. Although this could take longer to compute, it is expected that the effort will be compensated by doing re-planning less frequently. We believe that avoiding over-commitment is a key issue in dynamic environments.

- **Distribution over time:** The controller should be opportunistic and distribute motion planning over time. Priority should be given to the most urgent computations (e.g., the generation of the paths to execute next), but, at any instant, if there are some extra free computing resources, these should be used to compute future paths according to what is most likely to happen. Distribution of problems over time implies that the controller be able to decide to start an operation, even if it has not been fully planned. For instance, consider a robot arm that just received the urgent task to unload a part from a conveyor. The controller may plan a path to grasp the part, execute the path, and, while executing this path, plan the path for transferring the part from the conveyor to its storage area. In order to do so, the controller must anticipate that while it will be executing the first path, some computing resources will become free, hence available to plan the next path. This implies that the controller should have the ability to plan its own activities and assign computing resources to them.

Our future research will investigate the distribution of path-planning computation along each individual axis. We will also design a software architecture allowing computation distribution along several axes simultaneously. We have started investigating the distribution of path planning computation along several of the above directions. We describe our specific work in more detail in the next section.

4.2 Parallelization of Path Planning

The use of interconnected parallel computers for the computationally intensive path planning problem is one axis along which we pursue fast solutions for this problem. Parallel computers provide more computational power and use effectively more memory than current uniprocessors. In addition, they implement various programming models (shared memory, message passing or the data parallel model). Each of these models reflects a different programming philosophy and offers different capabilities to the user. We plan to adapt some of the existing path planning algorithms (especially the Randomized Path Planner) to exploit the capabilities of parallel architectures and obtain significant speedups for these algorithms. At a later stage, it might be possible to devise inherently parallel path planning algorithms that are designed directly for parallel machines and exhibit better performance than parallelized versions of existing algorithms.

We have started our research with the parallelization of RPP. This planner, although one of the fastest of its kind, is still too slow to be reasonably used in a dynamic robot motion planning environment.

There are three main parts in the RPP algorithm[2]. The first part is a preprocessing stage which involves the construction of two-dimensional or three-dimensional workspace potential fields. These potentials are later used to realize the configuration-space potential field function which is a function with a global minimum at the goal position of the robot. In the preprocessing stage, the configuration space obstacles can also be computed and stored, (if memory permits) in the form of a bitmap. Such a bitmap is used to reduce collision checking to an $O(1)$ operation and improve the performance of the planner considerably. The second part of the algorithm plans the path of the robot from an initial position to a goal position and is the most time consuming part. The path of the robot is found by searching for the global minimum of the potential field function. The third part smooths the produced path to avoid unnecessary motions.

Various forms of parallelization are possible for the RPP. Some could involve medium-size multiprocessor. Others could make use of fewer, strongly interconnected, powerful processors. e.g. a small-scale shared memory multiprocessor or even a network of available workstations. We plan to mainly focus on the parallelization of RPP, (1) on a small-scale shared memory multiprocessor (namely a Silicon Graphics SGI 4D/240 with 4 processors and 32 Mbytes of physical shared memory), and (2) on a network of UNIX-based workstations (actually, the network of workstations available in our laboratory). While the first alternative is likely to bring more significant improvements, the second has the advantage of using widely available computing resources.

All three parts of the algorithm exhibit interesting potential for coarse-grain parallelism. In the preprocessing stage, the construction of the potential field and the configuration space bitmap is worth parallelizing because these constructions are time consuming, especially in the three-dimensional case. Smoothing can be done incrementally by a processor that happened to be idle at some point of the computation. It can also be achieved in the end with a truly parallel algorithm, possibly different from the one used now.

The second part of the RPP is where we believe we should focus most of our efforts since it is the most time consuming. The planner produces the robot path by following the negative gradient of the potential field until the path reaches a local minimum of the potential field function. At this point, random motions are initiated until the local minimum is escaped. We would like to reduce the time spent in escaping the local minima of the potential field by introducing parallelism in two different ways. They are summarized below:

1. When stuck at a local minimum, we can let every available processor execute its own random motion, and then a down motion following the negative gradient of the potential field function until a new minimum is attained. When one of the processors finds a local minimum with lower potential than the initial one, it informs all other processors, which in turn stop the random motions they were executing. After that, all available processors start executing random motions to escape the new local minimum of the potential field function. And so on.

[2] See J. Barraquand and J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *The International Journal of Robotics Research, MIT Press*, 10(6), Dec. 1991, pp. 628–649.

2. Another way to increase our chances of escaping a local minimum is to have one available processor execute a rather long random motion, while the other processors perform down motions following the negative gradient of the potential field from various initial positions on the random path produced by the first processor. When a minimum with a lower potential is discovered, all processors are informed. They all terminate their jobs and start working to escape the new minimum.

The first option for parallelism described above offers a reasonable task size for coarse-grain parallelization. The synchronization overhead is small compared to the computational work (random and down motions) performed by each of the processors. In addition, the tasks assigned to the different processors are independent and communication among the processors used occurs only when one of them has found a better local minimum. The second option for parallelism involves more complicated communication. A combination of the two options for parallelism is also possible.

In the coming quarters we plan to develop the first option and to experiment with it.

4.3 New Methods for Fast Path Planning

For robots with few degrees of freedom, several efficient planning methods based on different principles are available. They make it possible to study the distribution of planning computation over planning methods. However, for robots with more degrees of freedom, the only reasonably efficient and reliable method known today is RPP. Although the parallel implementations of the RPP under development should provide significant speedup relative to the current uniprocessor implementation, the basic algorithm remains the same and we know that it works rather poorly for some pathological cases. Parallelism will not eliminate these cases.

For this reason, we have initialized new research aimed at both improving the intrinsic performance of the RPP (by explicitly dealing with identified pathological cases, when possible) and proposing new faster algorithms (with hopefully different pathological cases). New interesting methods arising from this line of research will be combined with the RPP (distribution over planning methods; see Section 2).

4.4 Optimal-Time Multi-Arm Manipulation Planning

Here, we investigate three major issues:

- Manipulation planning, i.e., the generation of series of paths for manipulator arms separated by grasp/ungrasp/regrasp operations on movable objects. This a major task on which we want want to study.

- Coordination of multiple arms, e.g., cooperative displacement of the same bulky object, coordinated motion where one arm handles a movable object to another (to facilitate collision avoidance or deal with mechanical limits in one arm), parallel execution of a task.

- Optimal-time path planning, i.e., the generation of paths that can be efficiently executed, given the dynamic characteristics of the arms and the moved objects.

These issues relate to several aspects of the distribution architecture we seek to develop. Firstly, cooperative manipulation tasks is one of the domain in which we want to demonstrate our results. Secondly, manipulation plans are complicated enough to justify distribution of planning over time (see Section 2). Optimal-time planning yields information about how long it will take to execute motions. This information is critical to study distribution over time. Thirdly, fast planners delivering the best on-line service possible makes real sense only if the generated plans are reasonably efficient.

We have started research in connection with our efforts towards optimizing the motions found by the RPP. We have built an RPP-based that planner succeeds in getting a movable object to a given desired location in a collision-free manner, but, despite the postprocessing smoothing phase, the trajectory is far from optimal due to the insertion of random motions utilized in the planning. The path needs to be treated further. We propose using a time-optimal control algorithm to smooth the path to one that the robot can follow in minimum time. Furthermore, we obtain a path that is optimally time parameterized, that is we have a trajectory that can be sent to the robot controller for actual execution.

Bobrow et al. have developed a time optimal control algorithm for an open chain robot following a specified path.[3] Using this algorithm to set the cost of a given path (minimum time of travel) the path can then be perturbed in the direction of the negative gradient of this cost to some locally minimum solution. The resulting path is one that the robot can follow to the goal in locally minimum time. This would be the desired smoothing of robot motions mentioned previously. However, manipulation paths consist of closed chain robot motions (multi-arms grasping the same object). We are currently working to extend the Bobrow algorithm to find time optimal motions for closed chain robot systems. One can then use this modified algorithm as a cost function in the proposed smoothing optimization of manipulation paths.

4.5 Experiments in Manipulation Planning

Our research on distributed motion planning will yield a set of software modules running under the ControlShell system to control actual manipulator robots available in the ARL (Aerospace Robotics Laboratory at Stanford). Our goal is to demonstrate manipulation of multiple movable objects by two cooperating arms in an uncertain, dynamic environment.

We started working on a first integrated demonstration using a manipulation planner that we devel-

[3]See J.E. Bobrow, S. Dubowsky, and J.S. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, MIT Press, 4(3), 1985.

oped over the past year.[4] This planner generates synchronized paths of two arms for manipulating movable objects. It also inserts grasp/ungrasp/regrasp operations between paths. Currently, it runs off-line. Our first task will be to make it run on-line under ControShell, assuming a static environment.

This investigation requires distributing the code in the ControlShell environment and using various interface standards appropriately. To facilitate present and future transfer of code between our laboratory and ARL, we started developing a simulator which will allow us to check that some major constraints are satisfied, before actually transferring the code.

4.6 Landmark-Based Mobile Robot Navigation

Contingencies during the execution of a robot motion plan can be classified into two broad categories: systematic and random. Systematic are all those uncertainties in the behavior of the robot or in the modelling of the workspace, which can be fitted into a "meaningful" stochastic model at planning time. Random is everything else. The significance of the existence of an uncertainty model at planning time is that we can use it to build plans, that are robust with respect to the modelled uncertainties. A robot can plan ahead for the most probable contingencies, and if they happen, it will be able to adjust its behavior in order to achieve its goal without having to replan. Thus, we can use this technique to trade planning time for swifter and more secure execution. This does not exclude, however, some form of reactive re-planning when a non-modelled event occurs at execution time. Actually, reaction plan planning (based on some prior model of uncertainty) and reactive replanning (based on error detection at execution time) should cohabit in a robust mobile robot navigation system (see Section 2: distribution of commitment over time). In this section we describe work focusing on reaction plan planning, i.e., planning with modelled uncertainty.

Uncertainties that can be modelled to be used with the aforementioned approach are control and sensing uncertainty, and to a lesser extent uncertainty in the geometric model of the workspace. The method of choice for creating plans that can deal with these kinds of uncertainty has been the *preimage backchaining*[5]. Unfortunately, it has been shown that under some general assumptions this method results in exponential algorithms, which cannot be used in practical applications.

Our approach is to restrict the class of problems under consideration, so that we can build polynomial algorithms that solve them. In order to do so, we started our research with a very simple model and designed a polynomial algorithm that solves it.[6] The current model is probably too simple to be practical, so we need to identify extensions to make it reflect the real world more accurately, and adapt our polynomial algorithm to handle them if possible. Ultimately, we expect to design a model that is complicated enough to describe the workspace and the robot satisfactorily, but still accepts an efficient polynomial-time motion planner.

[4]This work was done under DARPA contract DAAA21-89-C0002.

[5]See T. Lozano-Pérez. M. Mason, and R. Taylor, "Automatic Synthesis of Fine-Motion Strategies for Robots," *The International Journal of Robotics Research*, MIT Press, 3(1), 1984, pp. 65–96.

[6]Part of this work was started under DARPA contract DAAA21-89-C0002.

During this quarter we defined the initial simple model, designed a polynomial complete algorithm that can produce guaranteed plans in this environment, and completed its implementation.

The initial model is as follows: A point robot moves in a two-dimensional workspace free of obstacles. In the workspace there are scattered landmarks (distinguishable features of the workspace). Each landmark defines a circular region in the workspace, called its area of influence. The robot is equipped with sensors that can only sense a landmark whenever the robot lies within the corresponding area. Then the sensors provide to the robot perfect knowledge of its position. If the robot is outside all landmarks areas it has no sensing whatsoever. The robot utilizes two control modes, the perfect and the imperfect modes. The robot has perfect control only when it can sense a landmark. Imperfect control, which can be used anywhere in the workspace, prescribes that the robot will move in a cone about the commanded velocity direction. The half-angle of this cone is the directional uncertainty. The robot has no sense of time, so the modulus of its velocity is irrelevant. Initially, the robot lies in some area of the workspace. The goal of the robot is to reach and stop in another area of the workspace, the goal.

The algorithm first finds all landmark areas that overlap the goal directly or through other landmark areas, and designates them as the first subgoal (kernel). Clearly, if the robot attains this region, it can reliably get to the goal using perfect control. Additionally, the robot cannot terminate any motion outside a landmark area, because simply it does not sense anything else. Then the algorithm computes the non-directional preimage of the kernel trying to find out whether the initial-position region is included in it for the same commanded velocity direction. If yes, we return this direction. If not, we add to the kernel all the non-kernel landmark disks that intersect the non-directional preimage, and repeat the process until either the initial position is included in some preimage, or no more non-kernel disks are cut by the preimage. In the first case we return success and the corresponding plan; in the second case we return failure, but we can still return the part of the plan that has been constructed. Unfortunately, in case of failure we cannot provide a guaranteed initial command to the robot. Nevertheless, we can let the robot execute an initial Brownian motion with reflection on the boundaries of the workspace, until it enters a landmark region for which a guaranteed plan exists. From there on it can reach the goal safely. The probability of success of such a Brownian motion tends to one as time goes to infinity and the expected completion time shrinks as the termination area (the landmark areas for which we have found a complete plan) grows. From the above, it can be seen that a plan is produced in the form of reaction rules. The robot reacts to its entrance into a particular landmark area in a prespecified way. This plan structure can be also useful in case an unexpected event causes the failure of a "guaranteed" motion command. The robot can again wander a bit in the workspace until it reconnects itself with the guaranteed plan.

Our implementation focused of course on preserving polynomiality. For the computation of the non-directional preimage we first use a classical planar line-sweep algorithm to compute a directional preimage for an arbitrary velocity direction, and see whether it includes any initial position disks or cuts any non-kernel landmark disks. Then realizing that as the velocity direction rotates the topology of the preimage, as well as the set of initial position disks that are included in and the set of non-kernel landmark disks that are cut by it, change only at a polynomial number of critical orientations, we only update (or sometimes recompute) the preimage at these critical orientations.

The complexity of the implemented algorithm is $O(\ell^3 \log \ell)$, where ℓ is the number of landmarks. We implemented it in C on a UNIX-based DEC-5000 workstation, and it runs reasonably fast even for complex environments (2 seconds up to 2 minutes for 50 disks and various values of the directional uncertainty).

In the coming quarters, we plan to investigate several extensions of this algorithm. The first extension will consist of adding obstacles.

4.7 Mobile Robot Navigation Toolkits

Developing a reliable, integrated mobile robot navigation system from scratch is a very time consuming task. This is because a mobile robot navigation system consists of many interacting components. However, many of these components are basic and common among various navigation systems. Therefore, there is no need for "re-inventing the wheel," whenever a system is implemented to satisfy the needs of a new application. The key idea therefore is to develop and maintain reusable software modules, which we call toolkits. These toolkits will be used as tools and building blocks for constructing various mobile robot experimentation systems. Some of them will be based on pre-existing methods. Others will require specific additional research.

During this quarter, we have started designing the layout of the toolkits and identifying major functions to be accomplished by the toolkit modules. The toolkits will be organized into three layers:

1. **The Geometry Engine Layer:** This layer provides the common functions required by the various toolkits in the Building Blocks Layer. We are implementing a first version of the Geometry Engine consisting of the basic 2D geometric reasoning functions such as line and curve intersection, polygon intersection and clipping, which are common to several toolkit modules.

2. **The Building Blocks Layer:** This layer will consist of various toolkits that would support the development of the different components of a mobile robot navigation system. Currently we are focusing on the development of the following toolkits:

   - Path Tracking Toolkit.
   - Sensor-Based Localization Toolkit.
   - Map Building Toolkit.
   - Motion Planning Toolkit.

3. **The System Developer Layer:** This layer will consist of various structures to help organize the components from the different toolkits into a reliable navigation system.

Using the building blocks and tools provided by the Toolkits, we expect to develop several mobile robot experimentation systems.

4.8 Simulator for Multiple Mobile Robots

Although experimenting with real robots is needed to conduct realistic research with mobile robots, it often turns into a frustrating and painful experience. Availability of powerful toolkits, like those mentioned above, can certainly increase productivity of robot software development. Nevertheless, there still remains the difficulty of debugging large integrated navigation systems. This is especially true when the environment is dynamic and incompletely known, and when multiple robots are involved. The two traditional ways of programming robots - on-board programming and cross-compile-down-loading - are then much too rudimentary.

To overcome these difficulties, we have started developing a graphic multi-robot simulator (with their sensors). Robot programs can alternatively run with this simulator or with real robots. To proceed faster in this work we have adopted the simulator provided in the UNIX-based Nomadic Host Software Development Environment. In this environment, the host computer controls the robot via a serial radio link using a simple serial communication protocol. In addition, the software environment provides a robot simulator which can greatly facilitate the testing and debugging of robot programs. We are working on modifying and extending the simulator and the interface to better fit our needs. In particular, we are currently investigating, in cooperation with Nomadic Technologies, the extension of the robot simulator to handle multiple robot simulation. This simulator will be able to run several robots concurrently, although each robot will be under the control of its own navigation system.

Chapter 5

Applications

It is not possible to develop generic technology without multiple, specific applications to test and refine the ideas and implementations. As such, we are actively seeking sites, both internally and externally to provide the compelling test beds that will make this project succeed. These driving applications span a variety of the most important target users: high-performance control, intelligent machine systems, underwater vehicle command and control, and remote teleoperation. Several of these projects will reach for new limits in advanced technology and system integration; others will address real-world problems in operational systems.

With the reduced funding levels, we will not have the resources to support all of the originally proposed technology evaluation sites. However, we believe these sites are crucial to the development of ControlShell into a viable technology for "real-world" use. Thus, we have actively pursued alternative means of supporting external sites. We have been successful in securing several new test applications. These sites will either function with minimal support, or fund their own support. This contract is funding only one directly—the robotic composite structure layup project.

This chapter highlights some of the activities of these projects. The highlights this quarter include:

- We have begun a research project into the use of ControlShell as a basis for an automated composite structure layup system.
- A new ControlShell application in ultra-precision machining has been started (externally funded).

The currently-active ControlShell applications are:

- Robotic Composite Layup, by Rick Hosey (joint project between the ARL and the Stanford Structures and Composites Laboratory; description provided below).
- Precision Machining, by The Stanford Quiet Hydraulics Laboratory. (Description provided below).

- Intelligent Machine Architectures, by Lockheed Missiles and Space Corporation.
- Remote Teleoperation, by Space Systems Loral Corporation.
- Space-based Mobile Robot Systems, by several ARL students (NASA-sponsored).
- High-Performance Control of Flexible Structures, by several ARL students (AFOSR-sponsored).

5.1 Robotic Composite Layup

This section describes an application of the software architectures, communications protocols and interfaces developed under this contract. This application explores a novel use of a distributed force/position control system.

Force/position control (i.e. control of end effector contact force normal to a constraint surface while controlling end effector position along the surface) is an important area of study in robotics. The force and position control problem has been studied in considerable depth for the case where the surface is fixed. This research will focus on force/position control for the case where the constraint surface is attached to a robot arm, and the force and relative position are generated by the end effector of a cooperating robot arm. Although the force/position control problem is an interesting problem by itself, this research is specifically aimed at developing a control system suitable for application in robotic fabrication of parts made of composite material.

Progress to Date This quarter the requirements for the force/position control system were defined. The requirements are generated from an actual application of robots in the manufacturing of parts made out of composite materials in a joint project between the Aerospace Robotics Lab and the Stanford Structures and Composites Lab (SSCL). SSCL is developing a technique for streamlining the manufacturing of composite parts. Composite parts are currently manufactured in two steps: layup and processing. In the layup step, sheets of composite material are placed on a tool plate or mold that has the form of the part. The layup step is followed by the processing step where heat and pressure are applied to consolidate the sheets of material into a single, rigid structure. The SSCL is developing a technique, called fiber placement, by which heat and pressure are applied at the point where the composite material is being placed on the tool plate; thus combining the layup step and the processing step into a single, integrated process.

Figure 5.1 is a schematic diagram of a robotic system that will fabricate composite parts by the method of fiber placement. A key feature of the system is the tape-laying head. This head dispenses a ribbon of material comprising carbon fiber embedded in a thermoplastic matrix. An important characteristic of the thermoplastic matrix is that it becomes pliable and sticky when it is heated, and it hardens as soon as it cools. These are ideal characteristics for manufacturing by fiber placement. As the tape- laying head dispenses the composite tape, hot nitrogen is blown out of the heater to heat the tape plus composite material that has been previously deposited on the mold.

The robot arm will move the head over the surface of the mold so that the wheel applies the correct pressure to consolidate the hot tape with the hot material on the mold. The quality of the finished part is determined by the temperature distribution and the stress distribution during manufacture. The speed that the head moves along the surface is an important factor in the determination of the temperature distribution. The force applied by the wheel is also important. If too little force is applied, then the composite will not be fully consolidated. If too much force is applied, then layers of composite that were previously applied will be ripped apart by the stresses. Since the quality of the finished part is determined by the speed of application and the force applied, accurate force/position control will be a crucial feature of a robotic manufacturing system based on this technique.

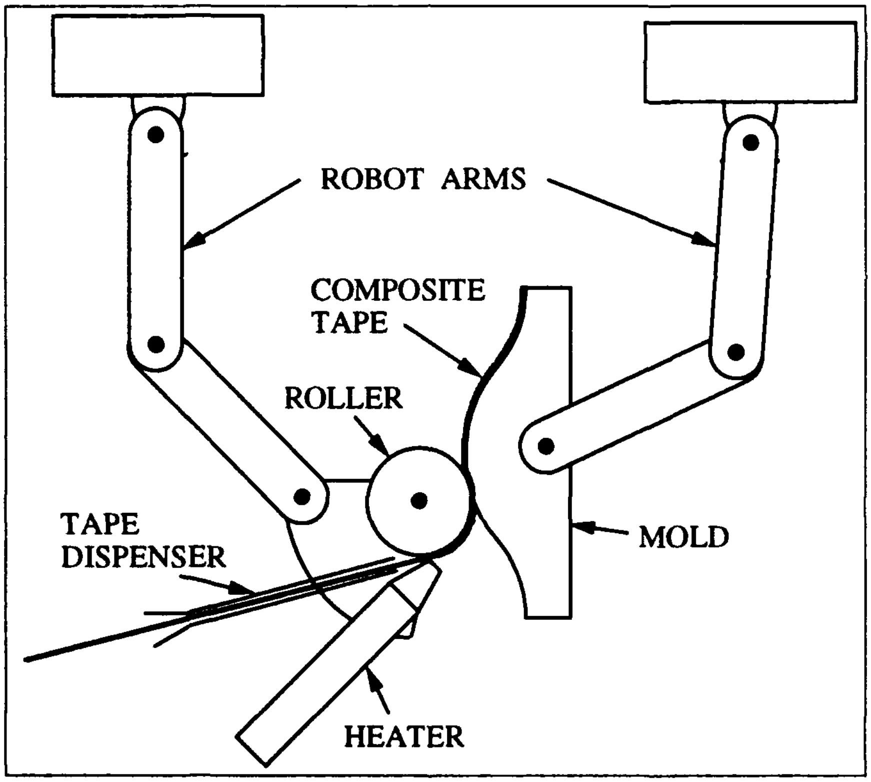

Figure 5.1: **Robotic Manufacturing System Schematic**

This schematic depicts the robotic manufacturing system based on cooperating arms.

Cooperating robot arms offer several advantages over a single robot arm. One advantage is that two robots with limited degrees of freedom may be combined to form a system with the capability for increased degrees of freedom of relative motion. This is illustrated by the application of composite material onto a dome shaped part. If the mold for the dome were fixed, a single arm would require at least five degrees of freedom (x,y,z displacement; yaw, pitch angle) to apply the material. However, the dome shaped part could be made by a three degree of freedom (x, y displacement; yaw angle) robot applying the composite in 90 degree arcs while a cooperating one degree of freedom (roll angle) rotates the mold about the axis of symmetry. Another advantage of cooperating robots is that the range of motion in a redundant degree of freedom is increased. For example, two robots with a y-axis motion range of 1 meter could be combined to produce a relative motion range of 2 meters. These potential performance gains motivate the study of force/position control between cooperating robot arms.

Results The following requirements for the robotic force/position control system were identified:

- Accurate force and position control are required to manufacture high quality parts.
- Cooperating robot arms will be utilized to obtain the advantages outlined above

5.2 Precision Machining

The Stanford Quiet Hydraulics Laboratory, located in the Durand building next to the ARL, will also utilize ControlShell to control an extremely high precision experimental lathe. A short description of this project is included here for reference.

Ultra precision machining requires the careful integration of several different technologies. Under the aegis of the Office of Naval Research, Stanford University began interdisciplinary research in novel optical metrology methods, laminar-flow based quiet hydraulic actuation, and computer modelling of machine tools. We are continuing this research with the support of an NSF Strategic Manufacturing Initiative grant (NSF-DDM-8914232) for research in ultra precision machining.

Although our research is focused on demonstrating these novel technologies on a small scale diamond turning machine, the technologies are applicable to any precision machine tool.

The purpose of the Quiet Hydraulics Laboratory is to develop and evaluate actuators, sensors, and controls for use in very high precision machine tools. We are building, using technology developed at Stanford, a diamond turning lathe. This lathe must achieve repeatability in figure and finish of 75 nm (3 microinches)—in other words, optical quality figure and finish. We have developed a family of quiet, laminar flow hydraulic actuators. In hydraulic actuators, the working fluid also carries away the heat generated by inefficiencies. Laminar flow devices are used in order to minimize pressure and flow fluctuations that might produce strain and seismic disturbances in the machine.

With the proper instrumentation, actuation, and control, it is possible to synchronize the tool motion with the spindle rotation on a lathe. One could then turn, on a lathe, a non-axisymmetric surface. An application might be the manufacture of a long focal length, off-axis optical surface, machined at a smaller radius. As an easily evaluated test of the system, we will machine a plane, with an optical quality finish, whose normal is inclined with respect to the spindle axis of rotation.

Pushing the boundaries of accepted limits in ultra precision machines requires integration of better actuators with better sensors and controls. Research in developing new technologies and research in the intelligent applications of new or different technologies are key to economic competitiveness. The adoption of more precise actuation in machine tools can have a revolutionary effect on the US manufacturing base.

This project is led by Hy Tran, under the guidance of Professor Daniel Debra.

We are pleased to have this project as a driving ControlShell application.